A Permissive Graphical Patcher for SuperCollider Synths

Frédéric Dufeu
CeReNeM
University of Huddersfield
f.dufeu@hud.ac.uk

ABSTRACT

This article presents the first version of a permissive graphical patcher (referred to in the text as SCPGP) dedicated to fluid interconnection and control of SuperCollider Synths. With SCPGP, the user programs her/his SynthDefs normally as code in the SuperCollider environment, along with a minimal amount of additional information on these SynthDefs, and programs Patterns according to a simple SuperCollider-compliant syntax. From the execution of this SuperCollider session, the SCPGP interface allows for the definition of higher-level Units, composed of one or several SynthDefs. These Units can then be used in the graphical patcher itself, where the user can easily create graphs of Units, set their parameters, and, where applicable, assign them Buffers and Patterns. Permissiveness is a key principle of SCPGP: once SynthDefs have been successively tested as valid SuperCollider code, the user must be able to interconnect them with no limitation regarding connector properties (signal rate, number of channels) or the order of execution on the SuperCollider tree of Nodes. SCPGP offers a range of flexible patching operations, to foster a fully fluid and open-ended experimentation from a network of user-defined SuperCollider Synths.

1. INTRODUCTION

The variety of creative uses of SuperCollider, described by its authors as "a programming language for real time audio synthesis and algorithmic composition [1]", is assessed by its initial developer, James McCartney, in the foreword to *The SuperCollider Book*. "With SuperCollider, one can create many things: very long or infinitely long pieces, infinite variations of structure or surface detail, algorithmic mass production of synthesis voices, sonification of empirical data or mathematical formulas, to name a few. It has also been used as a vehicle for live coding and networked performances [2, p. IX]".

SuperCollider has a client-server architecture: the server application, scsynth, performs the audio synthesis and processing. Its client, sclang, is the interpreter for the SuperCollider programming language itself, and sends OSC messages to the audio server. A canonical use of SuperCollider is to write code in sclang and execute it to command the DSP operations performed by scsynth. On the one hand, SuperCollider can be used as a primarily text-based creative environment, and features such as the Just-in-Time library (JITlib) [3] offer an extended flexibility for coding-driven live performances. On the other hand, sclang has a range of Graphical User Interface (GUI) features, allowing for advanced non-text-based user interactions with both sclang and scsynth [4].

The development of the graphical patcher presented in this article is motivated by one of the possible uses of SuperCollider: an advanced text-based design of personalised DSP engines (synthesizers, samplers, processers, typically expressed as SynthDef objects in the SuperCollider language), followed with modular interconnections of these engines. Widespread equivalents to the second part of this approach in the physical world are the assemblage of modular synthesizers or the combination of guitar effect pedals. Although such interconnections can be operated solely by coding, it is here assumed that in a situation involving a number of modules contributing to a global audio graph, designating one particular module, regardless of the operation to perform on it, is easier and quicker if this module is represented as a graphical object on a two-dimensional visual workspace than as a variable name in a textual environment[1].

Beyond the ability to interconnect graphically SuperCollider-designed DSP modules, the essential advantages of implementing a patcher using the sclang/scsynth couple as a backend, as opposed to programming directly in visual languages such as Max or Pd, are twofold. First, the large library of Patterns delivered with SuperCollider [7] enables to create Event-driven Synths with great flexibility and expressivity: simple or complex Patterns can control the evolution of all the parameters of a given module, including its Buffer references and input and output Buses, providing an extended dynamism to the global DSP graph. Secondly, implementing a GUI that is extrinsic to the considered programming language fa-

[1] More generally, the pros and cons of the textual and graphical computing paradigms are highly dependent on their contexts of use. In an article on OpenMusic published in the *Journal of Visual Languages and Computing*, Jean Bresson and Jean-Louis Giavitto affirm that "visual languages make programming and the access to computer resources more productive and useful to certain user communities, willing to design complex processes but not necessarily attracted to or skilled in traditional textual programming. They are supposed to ease programming activities (e.g. limiting syntactic errors), but also contribute to a more interactive relation between the user and the programs [5, p. 364]". Bresson and Giavitto reckon that "this idea can be argued" and point in particular to one empirical study, out of the scope of creative computing [6].

vours the design of patching operations that go beyond what the visual programming languages for musical creation normally permit, thus facilitating studio experimentation and performance expressivity. The body of this article presents the global architecture of the proposed environment, before developing the features and uses of the SuperCollider code in this context, and of the two main workspaces of the GUI application itself: the Unit Maker and the Unit Patcher.

2. OVERALL ARCHITECTURE

The permissive graphical patcher for SuperCollider, referred to in this text as SCPGP for convenience and clarity[2], is being developed both in the SuperCollider language itself and as a separate GUI application built from JavaScript code in Max, embedded in a JSUI (JavaScript User Interface) object including the MGraphics library[3]. The overarching principle of SCPGP is that its user should design both elementary DSP modules and Patterns for the control of dynamic Synths as code in the SuperCollider language, and that everything else – assembling Units and playing with them – should be done from the graphical interface.

Using SCPGP first requires executing a SuperCollider document, here referred to as the SuperCollider session, that contains the user-defined SynthDefs and Patterns, as well as the backend algorithm responding to the user's actions from the graphical interface. Once the SuperCollider session has been executed, the GUI application can communicate with scsynth via sclang, with simple commands sent over a network with UDP (figure 1).

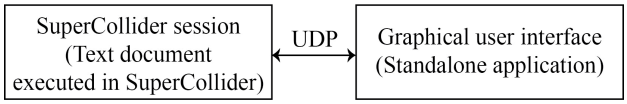

Figure 1. Overall structure of the SCPGP environment

From one given SuperCollider session (i.e., from one set of defined SynthDefs and Patterns), the user can create, save, and restore one or several GUI sessions. Creating a GUI session essentially dumps the SynthDef and Pattern information from SuperCollider to the GUI application, where the user can make her/his own Units and patch them together to generate sound and experiment.

One important aspect of the SCPGP design is that the user does not play by patching strictly SynthDef-based modules, but with Units that she/he must make from one or several of the SynthDefs declared in the SuperCollider session. The reason for this design is that some DSP modules cannot be sensibly conceived from only one SynthDef. For instance, a Unit supposed to read grains from an incoming audio stream requires the definition of two SynthDefs: the first SynthDef defines a template to record continuously the incoming audio stream into a Buffer, while the second SynthDef defines a template for reading a grain from that Buffer, with the desired parameters and envelope. A pattern can then trigger grains as dynamic and self-freeing instances of Synth that refer to the second SynthDef, while the recording Synth, referring to the first SynthDef, remains permanently active from the creation to the destruction of the unit. Therefore, the GUI application of SCPGP is itself constituted with two distinct and non-simultaneous workspaces: the Unit Maker, in which the user chooses SynthDefs and assembles them into Units, and the Unit Patcher, in which the user actually generates sound by patching together her/his previously defined Units, controls their parameters and, where appropriate, assigns them Buffers and Patterns. Figure 2 summarizes the workflow in SCPGP.

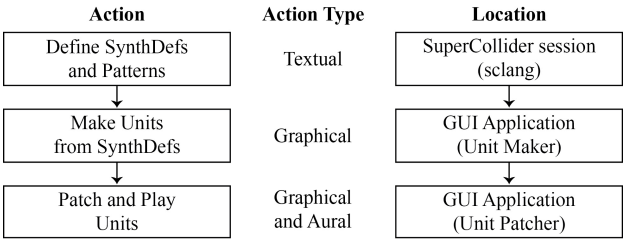

Figure 2. Workflow within SCPGP

3. THE SUPERCOLLIDER SESSION

The SuperCollider session is a text document read by the SuperCollider IDE. At the top of the document are the user definitions: essentially, SynthDefs and, if needed, Patterns. Some general settings can also be edited in that part of the document[4]. The rest of the text document should not be edited: it contains the algorithm responding to the user's actions from the GUI application. Some aspects of its implementation are described in the paragraphs dedicated to the Unit Maker and the Unit Patcher; in this paragraph are explained the declaration modes for SynthDefs and Patterns.

3.1 SynthDef declaration

In the SuperCollider session, SynthDefs are declared sequentially within a function named "func_defineUserSynthDefs" (figure 3). For each SynthDef, the user must provide a unique name (e.g. below 'Stereo Dac', 'Loop Sampler'), then declare the SynthDef as he would normally do in SuperCollider, by calling on the SynthDef class the implicit "new" method, with the name of the SynthDef and the UGen graph function as arguments. The SynthDef is then added to the SynthDescLib[5] and sent to scsynth with the "add" method.

A function named "func_initSynthDefInfo" must then be evaluated with the name of the SynthDef as an argument: this function queries the SynthDescLib to provide the SynthDef information needed by the GUI application of SCPGP[6]. As some information cannot be inferred from the SynthDescLib, the user must provide manually an

[2] The product name for SCPGP is yet to be chosen and will be given on the public release of its first beta version.

[3] A first prototype was presented by the author of this article in 2012 at the French annual computer music conference in Mons [8]. The graphical user interface was then developed with OpenGL in Max.

[4] For instance, the UDP addresses and ports for communication with the GUI application.

[5] The SynthDescLib is a library of descriptions of SynthDefs.

[6] The collected SynthDef information is: static or dynamic status (deduced from the "hasGate" member of the SynthDef description), inputs and outputs properties (audio or control rate, number of channels), argument names.

array of buffer references containing, for each reference, the control name for the buffer and its number of channels[7].

```
func_defineUserSynthDefs = {
    var all_synthdefs_info = [];
    var synthdefname;
    var synthdefinfo;

    /************************************************************/

    synthdefname = 'Stereo Dac';

    SynthDef(synthdefname, { |in0, amp = 1|
        var signal = In.ar(in0, 2) * amp;
        Out.ar(0, signal);
    }).add;

    synthdefinfo = func_initSynthDefInfo.value(synthdefname);
    synthdefinfo.bufferControls = [];

    all_synthdefs_info = all_synthdefs_info.add(synthdefinfo);

    /************************************************************/

    synthdefname = 'Loop Sampler';

    SynthDef(synthdefname, { |out0, bufnum, rate = 1|
        var signal = PlayBuf.ar(1, bufnum, BufRateScale.kr(bufnum) *
rate, 1, 0, 1, 0);
        Out.ar(out0, signal);
    }).add;

    synthdefinfo = func_initSynthDefInfo.value(synthdefname);
    synthdefinfo.bufferControls = [\bufnum, 1];

    all_synthdefs_info = all_synthdefs_info.add(synthdefinfo);

    /************************************************************/
```

Figure 3. SynthDef declaration in a SuperCollider session of SCPGP for a Dac and a simple Loop Sampler.

At this stage of the SCPGP workflow, it is the responsibility of the user to ensure that the SynthDef declaration is valid SuperCollider code, and that the bufferControls array is conform to the SynthDef UGen graph.

3.2 Pattern declaration

The Pattern declaration in SCPGP is more specific than the SynthDef declaration, and takes place in a function named "func_defineUserPatterns" (figure 4). Each Pattern is initialised with a function named "func_initPatternInfo" that takes a unique name as argument (e.g. below 'Grains 1', 'Play Sample Once'). The user can then write sub-Patterns as members of the "pattern_info" dictionary[8]: first, a sequence for the durations of successive Pattern Events ('dur'); then, sequences for input and output Buses, and, where appropriate, for Buffer references; finally, sequences for Synth parameters.

At this point, the Patterns for input and output Buses and for Buffers do not take actual Bus and Buffer objects lists as arguments: rather, they take abstract indexes that will be updated when called from the GUI application. In the example of the 'Grains 1' pattern in figure 4, the Pattern design assumes that the Unit referring to this pattern can receive its input signal ('in0') from two different buses (0, 1) and send its output signal ('out0') to three different buses (0, 1, 2). When played, the Pattern will generate a succession of grains receiving signals from and sending to the actual Unit buses as follows: 1 (in: 0, out: 0), 2 (in: 1, out: 1), 3: (in: 0, out: 2), 4: (in: 1, out: 0), 5: (in: 0, out: 1), 6 (in: 1, out: 2). The same principle applies to buffer references. The Patterns for durations and parameters take lists of actual parametric values.

```
func_defineUserPatterns = {
    var all_patterns_info = [];
    var pattern_info;

    /************************************************************/

    pattern_info = func_initPatternInfo.value('Grains 1');
    pattern_info.type = \on;
    pattern_info.dur = Pseq([0.5, 0.1, 0.2, 0.3], inf);
    pattern_info.buses.in0 = Pseq([0, 1], inf);
    pattern_info.buses.out0 = Pseq([0, 1, 2], inf);
    pattern_info.parameters.freq = Pseq([440, 550], inf);
    pattern_info.parameters.len = Prand([0.5, 0.6, 0.7], inf);
    pattern_info.parameters.amp = Prand([0.1, 0.2, 0.3, 0.4], inf);

    all_patterns_info = all_patterns_info.add(pattern_info);

    /************************************************************/

    pattern_info = func_initPatternInfo.value('Play Sample Once');
    pattern_info.type = \on;
    pattern_info.dur = Pseq([0], 1);
    pattern_info.buses.out0 = Prand([0, 1]);
    pattern_info.buffers.bufnum0 = Prand([0, 1, 2]);
    pattern_info.parameters.rate = 1;

    all_patterns_info = all_patterns_info.add(pattern_info);

    /************************************************************/
```

Figure 4. Pattern declaration in a SuperCollider session

When a user-defined Pattern is called from the GUI application of SCPGP, the Pattern information is used to construct and interpret a Pbind object that can then be used to play the appropriate Synth with the actual Buses, Buffers, and parameters of the designated Unit.

Here again, it is the responsibility of the user to ensure that the Pattern declaration is valid SuperCollider code. When ready with SynthDef and Pattern declarations, the user can execute the whole SuperCollider session document to interpret its code. From now on, all user actions take place in the GUI application.

4. THE UNIT MAKER

4.1 Creation of a GUI session

If no GUI session has been previously created and saved, the user must create a new GUI session from the Unit Maker. This action simply asks SuperCollider to dump to the GUI application its SynthDef and Pattern information. The left sidebar of the Unit Maker is then populated with graphical representations of template Inputs, Outputs and SynthDefs (under forms visible in figure 5a below). These templates are the constitutive elements of a new Unit[9].

Each SynthDef is represented with its name as specified in the SuperCollider session. At the top of the rectangle are its inputs: red rectangles are audio rate inputs; orange rectangles are control rate inputs. At the bottom of the rectangle are its outputs. The width of the inputs and outputs represent their number of channels. The distinction

[7] In the example of figure 3, the bufferControls array is empty for the 'Stereo Dac' SynthDef, that has no buffer reference in its UGen graph function, and contains one control name ('bufnum') referring to a one-channel buffer for the 'Loop Sampler' SynthDef.

[8] In figure 4, Patterns are represented with the Pseq and Prand classes.

[9] As there is typically a large number of template SynthDefs in a session, the user can also display the template items as a standard text tree and create her/his own categories of SynthDefs to navigate more conveniently.

between static and dynamic SynthDefs is apparent in figure 5a. Static SynthDefs ('Buffer Recorder', 'Reverberation', 'Filter') are those that are permanent from the creation to the destruction of the parent Unit; dynamic SynthDefs ('Granulator 1') are those that are Event-driven from Patterns. Each input and output of a dynamic SynthDef can have any number of, respectively, virtual inputs and virtual outputs, so that the driving Pattern can receive from and send to different Buses, as mentioned in paragraph 3.2. Virtual connectors are displayed at the edge of a tree originating in the SynthDef connector.

4.2 Edition of a Unit graph

After creating a new Unit, the user can design its graph by clicking and dragging template Inputs, Outputs, and SynthDefs from the left sidebar of the Unit Maker to the central workspace[10]. Standard mouse and modifiers configurations facilitate a fluid patching[11]. Figure 5 shows an example of a Unit graph (figure 5a) and the representation of the corresponding Unit as to be used in the Unit Patcher (figure 5b). Inlets and Outlets are numbered automatically according to their left-to-right order; likewise, SynthDefs are labelled with their relative order of execution on the SuperCollider tree of Nodes.

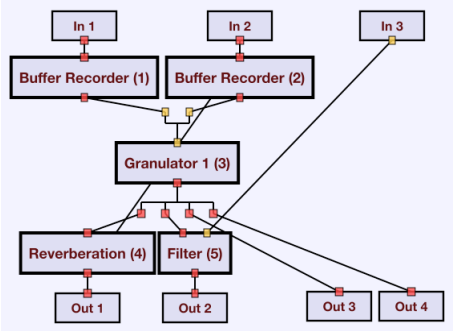

Figure 5a. A Unit graph with 3 Inputs, 4 Outputs, 4 static SynthDefs, and 1 dynamic SynthDef.

Figure 5b. The graphical representation of the resulting Unit, as to be used in the Unit Patcher

Figure 5a reveals permissiveness as one essential principle of SCPGP. Nothing prevents the user to patch any output into any input. An audio rate (red) output can send signal into a control rate (orange) input and conversely; a stereo output can be patched directly to a 4-channel input[12]; an audio output can be connected to the audio input of a SynthDef that is not below the origin SynthDef in the DSP graph (leading to the implicit creation of a block-size feedbacker)[13]; and it is possible to have as many cords from one output or to one input as needed[14]. All the corresponding signal conversions, block-size delaying for feedback, and mixing are handled by implicit Synths, automatically compiled when the entering the Unit Patcher, and are entirely transparent to the user.

4.3 Edition of the bypasser graph of a Unit

In the Unit Patcher, all Units can be bypassed by ctrl-clicking them[15]. In the Unit Maker, the user can define a specific bypasser graph for Units. When going to the bypasser graph editor, the workspace shows the Unit graph with only its Inputs and Outputs. The user can then drag cords between those to define the Unit signal graph when bypassed.

4.4 Edition of the arguments of a Unit

The right sidebar of the Unit Maker displays the parameter arguments of the edited Unit. By default, these arguments are those of the static SynthDefs constituting the Unit, excluding those relative to Buses and to Buffers. The arguments of the dynamic SynthDefs are not displayed: they are to be handled entirely by Patterns.

In some cases, the user might want to map her/his own parameter names to the SynthDef arguments. A simple example of such a case is a Unit made of four parallel 'oscillator' SynthDefs, only taking 'frequency' as a parameter argument. Rather than having four 'frequency' arguments for the unit, it may be useful to only have one 'BaseFrequency' parameter and one 'Detune' parameter.

The Unit argument editor of the Unit Maker is a text-field in which the user can declare parameters as keywords, and then type formulas to map them to the low-level arguments. Following the example described above, the user can type the lines of text as in figure 6.

```
parameter BaseFrequency
parameter Detune
frequency[0] = BaseFrequency + (0 * Detune)
frequency[1] = BaseFrequency + (1 * Detune)
frequency[2] = BaseFrequency + (2 * Detune)
frequency[3] = BaseFrequency + (3 * Detune)
```

Figure 6. Example of argument mapping

In the Unit Patcher, each Unit of this type will then appear with value fields for 'BaseFrequency' and 'Detune'. Any formula that is valid SuperCollider code can be used

[10] As Units are in many cases derived from one single SynthDef, a button also enables the direct creation of a Unit from one SynthDef.

[11] These configurations enable: multiple object and/or cord selection (with shift), multiple selection with a selection rectangle, copy of selected objects (with alt), copy of selected objects with copy of the cords of the copied objects (with cmd/ctrl). Connecting the inlet of an object to the outlet of another or the same object, or conversely, is done by clicking on the first connector and clicking on the second connector. By cmd/ctrl-clicking and dragging the virtual connector of a dynamic SynthDef, the user can increase or decrease its number of virtual inputs or outputs. Renaming an object with an existing Input, Output, or SynthDef name replaces it with the corresponding object and maintains the patch cords.

[12] The built-in behavior of number of channels conversion in SCPGP depends on the ratio between the number of channels of the source and the number of channels of the destination. Should the user need a specific behavior, the SuperCollider algorithm is flexible enough to be changed with a minimal of amount of recoding. It is also possible to create SynthDefs with specific channel conversion behaviors and use them explicitly in the Unit graph.

[13] In figure 5a, the reverberation is fed back into one of the recorders.

[14] The SuperCollider design of SCPGP is such that one output writes to one unique bus whatever the number of destinations, but one input reads from several distinct buses (one bus per origin). Implicit mixer Synths are created when an input needs to read from more than one bus.

[15] Including Units with only inputs or Units with only outputs, for which there is no bypasser graph, and bypassing means muting.

for the mapping. The Unit argument editor also allows to set minimum, maximum, and default values, as well as user-readable names for all parameters.

All the information of the Unit Maker (Categorisation of Units and template SynthDefs, Unit graphs, bypasser graphs, edited argument) can be saved for later restoration from the GUI application[16]. When ready with the Units from a new or restored GUI session, the user can go to the Unit Patcher of the GUI application to patch and play her/his Units.

5. THE UNIT PATCHER

The Unit Patcher is where the user actually generates sound by sending commands to SuperCollider via the GUI application. When the Unit Maker configuration has been modified (i.e., some Units have been created and/or edited) and the user goes to the Unit Patcher, the GUI application dumps information on all its Units to the SuperCollider session. The SuperCollider algorithm then makes a database of Unit types, so that any command sent from the Unit Patcher is as efficient as possible for use in a real-time critical context.

5.1 Patching Units together

As for the Unit Maker, the left sidebar of the Unit Patcher is a template palette from which the user can drag and drop items to the central workspace. However, this sidebar only contains Units[17]: while patching, the user only considers Units, and SynthDefs are transparent. Apart from the type of handled objects, patching operations are identical in the Unit Patcher and in the Unit Maker (creation, selection, move, copy, deletion) and patching is entirely permissive: the output of a Unit can be patched into the input of any Unit, including itself, regardless of signal rates, numbers of channels, relative positions on the DSP graph, number of already incoming signals. The GUI application sends compact messages over the UDP network to the SuperCollider session, which then handles Group, Synth, and Bus creations, modifications, and deletions. For patching, these messages are as follows:

- createUnits [Unit type, position on SC graph];
- deleteUnits [Unit index];
- moveUnits [Unit index, new position on SC graph];
- createConnections [Origin Unit index, Origin Output index, Destination Unit index, Destination Input index];
- deleteConnections [Origin Unit index, Origin Output index, Destination Unit index, Destination Input index].

Each of these commands can take any number of arguments, and commands can be combined into one single message to SuperCollider. Therefore, the design of SCPGP has a built-in distinction between user actions in the Unit Patcher workspace and updates of the SuperCollider DSP graph. This enables the user to choose between two main patching modes: in the direct mode, the DSP graph is updated on each user action on the Unit Patcher (as would happen in Max or Pd). In the indirect mode, the user can perform several successive actions and only update the DSP graph with all modifications happening in one go by clicking an "update" button. This enables direct transitions between significantly different DSP scenes.

While the user acts solely upon Units and patch cords, the SuperCollider interprets the commands by handling the DSP tree reordering and the instantiation of transparent Synths and Buses: figure 7 shows a diagram of all the Synths created in SuperCollider (figure 7a) given the graph as seen by the user in the GUI application (figure 7b).

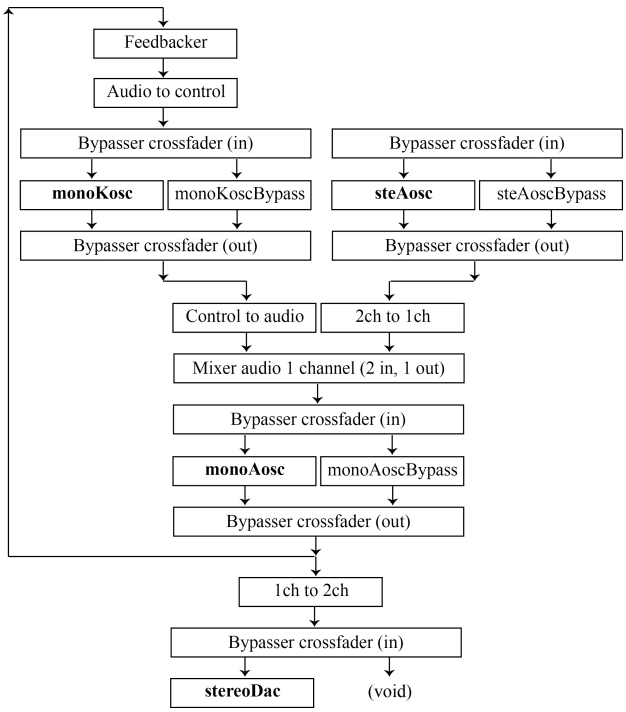

Figure 7a. A Unit graph as deployed in SuperCollider. Each box represents one Synth. In bold are the Synths corresponding to the core SynthDef of a given Unit.

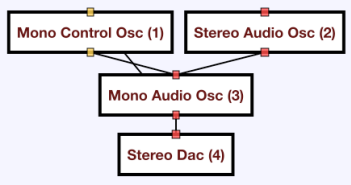

Figure 7b. The same Unit graph as defined by the user in the Unit Patcher

In addition to the direct and indirect modes of patching, some patching operations facilitate fluid changes in graph configurations. Those are especially useful when used in the direct patching mode, as they go beyond what is possible as one single action in standard visual programming environments for sound and music. These operations include:

- delete selected Units but maintain the cords going through them. In the example of figure 7b above, the user can for instance delete the "Mono Audio Osc" Unit – consecutively, the cords from "Mono Control Osc" and "Stereo Audio Osc" will be automatically repatched to

[16] The GUI session is saved as a JSON document. When restoring a session, SCPGP checks the restored SynthDef information against the SynthDefs of the SuperCollider session; if no mismatch is detected, the restored GUI session is validated and the user can go directly to the Unit Patcher.

[17] Along with user-defined categories of Units for display and navigation convenience.

"Stereo Dac", and the output of "Stereo Audio Osc" will also be patched into "Mono Control Osc".

- insert new or copied Units directly on an existing patch cord.

- rotate selected Units clockwise or counter-clockwise, maintaining patch cords. When the number of selected Units is two, this is a direct swap of both Units.

These operations are useful to change graph configurations quickly for experimentation in the studio, but are also enhancing performance expressivity: as McCartney stated in 2002, "The SuperCollider 3 Synth Server is a simple but flexible synthesis engine. While synthesis is running, new modules can be created, destroyed, and repatched, and sample buffers can be created and reallocated. Effects processes can be created and patched into a signal flow dynamically [9, p. 64]". The specific patching operations featured in SCPGP can benefit from the dynamism of scsynth: simple or complex modifications of the graph do not interrupt the signal processing, and can thus be used smoothly within a performance.

5.2 Handling Units on the graph

When a Unit has been created, the items of the right sidebar of the Unit Patcher enable a number of operations. The parameters, as defined in the Unit Maker, are modifiable via number boxes and are clipped between the user-defined minimum and maximum. As mentioned in paragraph 4.3, all Units can be bypassed by ctrl-clicking them on the Unit Patcher workspace.

Patterns are accessible as a list of names. They can be dragged and dropped onto the Pattern slot of a Unit's parameter panel: once the Pattern has been successfully checked against the internal dynamic SynthDef of the Unit, the user can simply play it and pause it with a toggle button. Here again, the GUI application is permissive. The user may have designed a Pattern with a specific SynthDef and Unit in mind, but in many cases the Pattern can be applied to another Unit that contains one or several dynamic Synths. Pattern values for existing parameter names will apply, values for non-existing parameters will simply be ignored. Parameters with no Pattern values will be played at their default values[18].

Buffers are not set in the SuperCollider session: they are allocated by the user from the Unit Patcher. There are two ways of allocating a Buffer from SCPGP: one is creating an empty Buffer by providing a number of channels and a duration in seconds, the other is to choose a sound file and fill a Buffer with it. Available Buffers can be simply dragged and dropped to a Unit's parameter panel for allocation to the appropriate Synth.

6. CONCLUSION

At the time of writing this article, the permissive graphical patcher for SuperCollider is fully functional regarding the features presented above, and is under internal alpha testing at the University of Huddersfield. The release and distribution of its first beta version will be publicly announced at the ICMC, on the presentation of this article. SCPGP offers great flexibility for those users who want both to design advanced DSP modules in SuperCollider and to interconnect them intuitively and fluidly into complex graphs. The environment can also be useful to beginners, who can focus on the UGen graph syntax of SynthDefs and adopt a modular approach immediately to test their Synths in different contexts, without having to write any code regarding Bus management.

Future work will consider user feedback following the first release of SCPGP, but two main threads are already under consideration. First, a Pattern editor will be developed in the GUI application itself: in the current state of SCPGP, Patterns cannot be modified after execution of the SuperCollider session. The Pattern editor will improve live flexibility for the control of dynamic Synths. Secondly, the implementation of a Unit Patcher scenario manager will enable the user to memorise particular patches and to navigate smoothly between her/his own previously defined DSP scenes.

[18] Here, permissiveness is increased if the user gives consistent names to the arguments of all SynthDefs (e.g. all 'freq' or all 'frequency', all 'amp' or all 'amplitude').

7. REFERENCES

[1] SuperCollider, software homepage on GitHub, available online at http://supercollider.github.io (retrieved May 11th, 2016).

[2] J. McCartney, "Foreword", in S. Wilson, D. Cottle, N. Collins (eds), *The SuperCollider Book*, The MIT Press, 2011, pp. IX-XI.

[3] J. Rohrhuber, A. de Campo, "Just-in-Time Programming", in S. Wilson, D. Cottle, N. Collins (eds), *The SuperCollider Book*, The MIT Press, 2011, pp. 207-236.

[4] T. Magnusson, "Interface Investigations", in S. Wilson, D. Cottle, N. Collins (eds), *The SuperCollider Book*, The MIT Press, 2011, pp. 613-628.

[5] J. Bresson, J.-L. Giavitto, "A Reactive Extension of the OpenMusic Visual Programming Language", *Journal of Visual Languages and Computing*, vol. 25, no. 4, 2014, pp. 363-375.

[6] K. N. Whitley, "Visual Programming Languages and the Empirical Evidence For and Against", *Journal of Visual Languages and Computing*, vol. 8, no. 1, 1997, pp. 109-142.

[7] R. Kuivila, "Events and Patterns", in S. Wilson, D. Cottle, N. Collins (eds), *The SuperCollider Book*, The MIT Press, 2011, pp. 179-205.

[8] F. Dufeu, "Une interface graphique de manipulation d'unités modulaires dans SuperCollider", *Proceedings of the 2012 Journées d'Informatique Musicale*, Mons, 2012, pp. 123-132.

[9] J. McCartney, "Rethinking the Computer Music Language: SuperCollider", *Computer Music Journal*, vol. 26, no. 4, 2002, pp. 61-68.